

BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

TRAINING CUỐI KỲ HỌC KỲ II NĂM HỌC 2023 – 2024



Sharing is learning



 **BAN HỌC TẬP**

Khoa Công nghệ Phần mềm

Trường Đại học Công nghệ Thông tin

Đại học Quốc gia thành phố Hồ Chí Minh

 **CONTACT**

bht.cnpm.uit@gmail.com

fb.com/bhtcnpm

fb.com/groups/bht.cnpm.uit

TRAINING

OOP – Cuối kỳ

⌚ **Thời gian:** 9h30 ngày 11/6/2024

📍 **Địa điểm:** B1.14

👤 **Trainers:** Nguyễn Lê Tuấn Anh – KTPM2023.1

Quách Vĩnh Cơ - KTPM2023.1



Sharing is learning

Các điểm kiến thức

1. **Lớp và đối tượng**
2. **Tính trừu tượng hóa và đóng gói**
3. **Phương thức thiết lập, phá hủy**
4. **Thành viên static, hàm bạn, lớp bạn**
5. **Nạp chồng toán tử**
6. **Kế thừa**
7. **Đa hình**



Sharing is learning

1. Lớp và đối tượng

- **Lớp** là một *mô tả trừu tượng* của nhóm các **đối tượng** cùng bản chất, ngược lại mỗi một **đối tượng** là một *thể hiện cụ thể* cho những **mô tả trừu tượng đó**.
- **Lớp** là cái ta thiết kế và lập trình
- **Đối tượng** là cái ta tạo (từ một lớp) tại thời gian chạy.



Sharing is learning

1. Lớp và đối tượng

Một lớp đối tượng bao gồm 2 thành phần chính:

- Thành phần dữ liệu (**data member**), hay còn được gọi là thuộc tính (**attribute**).
- Hàm thành phần (**member function**), còn có tên gọi khác là phương thức (**method**), là các hành động mà đối tượng của lớp có thể thực hiện.



Sharing is learning

1. Lớp và đối tượng

Khai báo Lớp:

```
class <tên_lớp>  
{  
    //Thành phần dữ liệu  
    //Thành phần xử lý  
};
```



Sharing is learning

1. Lớp và đối tượng

Khai báo và tạo đối tượng:

<tên_lớp><tên_đối_tượng>;

Gọi hàm thành phần của lớp

<tên_đối_tượng>.<tên_hàm_thành_phần>

(<danh sách các tham số nếu có>);

<tên_con_trỏ_đối_tượng>→<tên_hàm_thành_phần>

(<danh sách các tham số nếu có>);



Sharing is learning

1. Lớp và đối tượng

VD:

```
class A{  
    private:  
        int data;  
    public:  
        int getData();  
};  
int main(){  
    A a;  
    a.getData();  
}
```


Các điểm kiến thức

1. Lớp và đối tượng
2. **Tính trừu tượng hóa và đóng gói**
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình



Sharing is learning

2. Tính trừu tượng hóa và đóng gói

Tính trừu tượng hóa (Abstraction) là cách nhìn **khái quát hóa** về một tập các đối tượng có chung các đặc điểm được quan tâm (bỏ qua các chi tiết không cần thiết).

Trừu tượng hóa dữ liệu là một kỹ thuật lập trình và thiết kế dựa trên sự tách biệt của **Giao diện (Interface)** và **Thực thi (Implementation)**.

- **Giao diện** của một Lớp đối tượng là các hoạt động mà người dùng của một Lớp có thể thao tác trên các đối tượng của Lớp đó.
- **Phần Thực thi** bao gồm các dữ liệu thành viên (thuộc tính), phần định nghĩa của các phương thức.



Sharing is learning

2. Tính trừu tượng hóa và đóng gói

Đóng gói chính là quá trình ẩn đi **phần Thực thi** khỏi người dùng bên ngoài và giới hạn quyền truy cập vào nó. Người dùng của một Lớp chỉ có thể sử dụng **Giao diện** mà *không* có quyền truy cập vào **phần Thực thi**

=> **Tính đóng gói (Encapsulation)** là việc kết hợp các thuộc tính liên quan đến nhau cùng với một các phương thức hoạt động trên các thuộc tính đó, “**gói**” tất cả vào một trong lớp. Đóng gói để **che** một số thông tin và chi tiết cài đặt nội bộ để bên ngoài không nhìn thấy.



Sharing is learning

2. Tính trừu tượng hóa và đóng gói

Phạm vi truy xuất:

- Trong định nghĩa của lớp ta có thể xác định khả năng truy xuất thành phần của một lớp nào đó từ bên ngoài phạm vi lớp.
- **private**, **protected** và **public** là các từ khoá xác định phạm vi truy xuất
- Mọi thành phần được liệt kê trong phần **public** đều có thể truy xuất trong bất kỳ hàm nào.
- Những thành phần được liệt kê trong phần **private** chỉ được truy xuất bên trong phạm vi lớp.



Sharing is learning

2. Tính trừu tượng hóa và đóng gói

Ví dụ:

```
class TamGiac
{
    private:
        float a, b, c; //độ dài ba cạnh
        Int loaiTamGiac(); // cho biết kiểu của tam giác:
        float dienTich(); // tính diện tích của tam giác
    public:
        void Nhap(); // nhập vào độ dài ba cạnh
        void In(); // in ra các thông tin liên quan đến tam
        giác
};
```



Sharing is learning

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. **Phương thức thiết lập, phá hủy**
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình



Sharing is learning

3. Phương thức thiết lập, phá hủy

- Các phương thức thiết lập (**Constructors**): có nhiệm vụ thiết lập các thông tin ban đầu của các đối tượng thuộc về lớp đối tượng khi đối tượng được khai báo.
- Bất kỳ một đối tượng nào được khai báo đều phải sử dụng một hàm thiết lập để khởi tạo các giá trị thành phần của đối tượng.
- Hàm thiết lập được khai báo giống như một phương thức với tên phương thức trùng với tên lớp và không có giá trị trả về (kể cả void).
- Constructor phải có thuộc tính public.



Sharing is learning

3. Phương thức thiết lập, phá hủy

Phân biệt các loại **Constructor** :

Phương thức thiết lập mặc định (Default Constructors): là phương thức thiết lập các thông tin ban đầu cho đối tượng về lớp bằng những giá trị mặc định (**do người lập trình quyết định**).

```
PhanSo::PhanSo(){  
    tu    = 0;  
    mau   = 1;  
}
```



Sharing is learning

3. Phương thức thiết lập, phá hủy

Phân biệt các loại **Constructor** :

Phương thức thiết lập nhận tham số đầu vào (Parameterized Constructors): Là thiết lập thông tin ban đầu cho các đối tượng thông qua các tham số đầu vào. **Constructor** nhận tham số đầu vào có thể sử dụng để ép kiểu.

```
PhanSo::PhanSo(int a, int b){  
    tu = a;  
    mau = b;  
}
```



Sharing is learning

3. Phương thức thiết lập, phá hủy

Phân biệt các loại **Constructor** :

Phương thức thiết lập nhận tham số đầu vào (Parameterized Constructors): Là thiết lập thông tin ban đầu cho các đối tượng thông qua các tham số đầu vào. **Constructor** nhận tham số đầu vào có thể sử dụng để ép kiểu.

```
PhanSo::PhanSo(int a){  
    tu = a;  
    mau = 1;  
}
```



Sharing is learning

3. Phương thức thiết lập, phá hủy

Phân biệt các loại **Constructor** :

Phương thức thiết lập sao chép (Copy Constructors): là phương thức nhận tham số đầu vào là 1 đối tượng cùng thuộc lớp .

Thông tin ban đầu của đối tượng sẽ giống hoàn toàn thông tin đối tượng tham số đầu vào.

```
PhanSo::PhanSo(const PhanSo& a){  
    tu = a.tu;  
    mau = a.mau;  
}
```



Sharing is learning

3. Phương thức thiết lập, phá hủy

- Các phương thức phá hủy (**Destructor**): có nhiệm vụ thu hồi lại các tài nguyên cấp phát cho đối tượng khi đối tượng hết phạm vi hoạt động.
- Destructor, được gọi ngay trước khi một đối tượng bị thu hồi.
- Destructor thường được dùng để thực hiện việc dọn dẹp cần thiết trước khi một đối tượng bị hủy.
- Một lớp chỉ có **duy nhất** một Destructor
- Phương thức Destructor trùng tên với tên lớp nhưng có dấu ~ đặt trước
- Được tự động gọi thực hiện khi đối tượng hết phạm vi sử dụng.
- Destructor phải có thuộc tính public



Sharing is learning

3. Phương thức thiết lập, phá hủy

```
class Vector{
    private:
        int n;
        int* arr;
    public:
        Vector(int N = 0, int* Arr = nullptr): n(N), arr(Arr){}
        // Hàm thiết lập mặc định
        ~Vector(){
            n = 0;
            delete []arr;
            arr = nullptr;
        }
};
```

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình



Sharing is learning

4. Thành viên static, hàm bạn, lớp bạn

Thành viên static

- Các thành phần tĩnh (static member) là các thành phần thuộc về cả một **lớp** chứ **không thuộc về** một **đối tượng cụ thể**. Điều này có nghĩa là tất cả các đối tượng của một lớp đều chia sẻ chung một thành phần tĩnh, và nó có thể được truy cập mà không cần thông qua một đối tượng.
- Có hai loại thành phần tĩnh:
 - Các thuộc tính (dữ liệu thành viên) tĩnh (Static data member)
 - Hàm thành viên (phương thức) tĩnh (Static member function)
- Thuộc tính tĩnh **không phải là thuộc tính** của một đối tượng nào cả
- Một hàm thành viên tĩnh **không gắn với** bất kì đối tượng nào



4. Thành viên static, hàm bạn, lớp bạn

Khởi tạo static member

Muốn khởi tạo một thành viên tĩnh chúng ta sẽ thêm từ khóa static vào trước dòng khai báo của nó. Như các thành viên khác thì thành viên tĩnh cũng có thể được khai báo như là một thành phần **private**, **public** hay là **protected**.



Sharing is learning

4. Thành viên static, hàm bạn, lớp bạn

Khởi tạo static member

Ví dụ:

```
class HìnhChuNhat{  
private:  
    int width, length;  
    static int count;  
public:  
    void set(int w, int h);  
    int area();  
};
```



Sharing is learning

4. Thành viên static, hàm bạn, lớp bạn

Hàm bạn (Friend Function): không thuộc lớp nhưng vẫn có quyền truy cập vào thành phần **private, protected**.

VD:

```
class MyInt {  
private:  
    int data = 1;  
public:  
    friend int get(MyInt);  
};  
int get(MyInt a){  
    return a.Data;  
}
```



Sharing is learning

4. Thành viên static, hàm bạn, lớp bạn

Lớp bạn (Friend Class): là lớp có thể truy cập các thành phần **private**, **protected** của lớp xem nó là bạn.

VD:

```
class TOM{
    private:
        int cnt;
    public:
        friend class JERRY; // lớp bạn JERRY
};
class JERRY{
    public:
        void change(TOM T){
            T.cnt++;
        }
};
```

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. **Nạp chồng toán tử**
6. Kế thừa
7. Đa hình



Sharing is learning

5. Nạp chồng toán tử

- **Định nghĩa lại toán tử** đã có trên các **lớp đối tượng** do người dùng tự tạo để dễ dàng thể hiện các câu lệnh trong chương trình.
- Ví dụ:

```
PhanSo ps1(1,2);
```

```
PhanSo ps2(2,3);
```

```
PhanSo ketQua;
```

```
ketQua = ps1.Tong(ps2); // Dùng hàm
```

```
ketQua = ps1 + ps2; // Dùng overload operator
```



Sharing is learning

5. Nạp chồng toán tử

- **Bản chất:** Nạp chồng tức là định nghĩa các hàm có **cùng tên** nhưng **khác tham số** truyền vào.
- **Cú pháp:** Nạp chồng (overload) hàm có tên là "**operator@**", với @ là toán tử cần overload (+, -, *, /, ...).
- **Có 2 cách nạp chồng:**
 - Sử dụng phương thức của lớp.
 - Sử dụng hàm bên ngoài.



Sharing is learning

5.1. Nạp chồng bằng phương thức

- Cú pháp khai báo: `KDLTrảVề operator@(<DSThamSố>);`
- Ví dụ: câu lệnh `a+ b` sẽ tương đương với `a.operator+(b)`
- Số lượng tham số đầu vào = Số toán hạng – 1 (do toán hạng đầu tiên sẽ là **đối tượng gọi phương thức**).



Sharing is learning

5.1. Nạp chồng bằng phương thức

Ví dụ: Nạp chồng toán tử + bằng phương thức cho lớp `PhanSo`

```
class PhanSo {  
public:  
    PhanSo operator+(const PhanSo& ps);  
    //...  
};  
PhanSo::PhanSo operator+(const PhanSo& ps){  
    PhanSo kq;  
    kq.tu = this->tu*ps.mau + ps.tu*this->mau;  
    kq.mau = this->mau*ps.mau;  
    return kq;  
}
```



Sharing is learning

5.2. Nạp chồng bằng hàm bên ngoài

- Ví dụ: câu lệnh $a + b$ sẽ tương đương với `operator+(a,b)`
- Số lượng tham số đầu vào = Số toán hạng
- Hàm bên ngoài phải là **hàm bạn** để có thể truy cập vào các thuộc tính riêng tư của lớp.



Sharing is learning

5.2. Nạp chồng bằng hàm bên ngoài

- **Ví dụ:** Nạp chồng toán tử + bằng hàm bạn cho lớp **PhanSo**:

```
class PhanSo {  
    friend PhanSo operator+(const PhanSo& ps1, const PhanSo& ps2)  
        //...  
};  
PhanSo operator+(const PhanSo&ps1,const PhanSo& ps2) {  
    PhanSo kq;  
    kq.tu = ps1.tu*ps2.mau + ps1.mau*ps2.tu;  
    kq.mau = ps1.mau*ps2.mau;  
    return kq;  
}
```



Sharing is learning

5.3. Một vài lưu ý

- Các đối số truyền vào có thể là các **kiểu dữ liệu khác** nếu có phép **chuyển kiểu** tương ứng.

- Ví dụ:

```
// Chuyển kiểu bằng constructor
```

```
PhanSo(int a){  
    tu = a;  
    mau = 1;  
}
```

- Khi đó câu lệnh $a+5$ có thể được hiểu thành:
 $a.operator+(PhanSo(5))$



Sharing is learning

5.3. Một vài lưu ý

- Nạp chồng bằng phương thức của lớp thì **toán hạng đầu tiên** phải là **đối tượng** thuộc lớp đó.

- Ví dụ:

PhanSo a(2,3);

5 + a; // Sai

- Câu lệnh trên sai vì không thể gọi phương thức **operator+** trên số nguyên 5.



Sharing is learning

5.3. Một vài lưu ý

- Vấn đề có thể được giải quyết bằng cách sử dụng **hàm bạn** và các phép **chuyển kiểu** phù hợp.
- Lúc này câu lệnh $5 + a$ sẽ tương đương với:
`operator+(phanSo(5), a)`



Sharing is learning

5.4. Cách nạp chồng các toán tử cơ bản

- Toán tử số học
- Toán tử gán
- Toán tử so sánh
- Toán tử số học, gán kết hợp
- Toán tử tăng một, giảm một
- Toán tử nhập, xuất



Sổ tay kiến thức OOP



Sharing is learning

Các điểm kiến thức

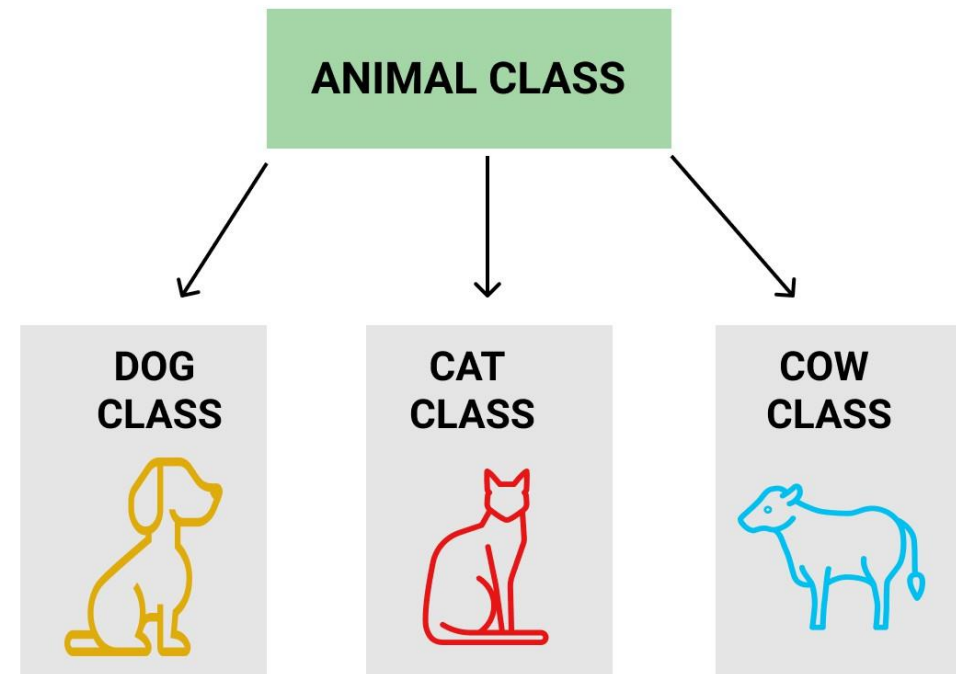
1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình



Sharing is learning

6. Kế thừa

- Một đặc điểm của ngôn ngữ lập trình dùng để biểu diễn mối quan hệ **đặc biệt hóa – tổng quát hóa** giữa các lớp (lớp này là trường hợp đặc biệt của lớp kia).



Sharing is learning

6. Kế thừa

- **Lợi ích** của kế thừa:
 - Kế thừa cho phép **xây dựng lớp mới** từ lớp đã có.
 - Kế thừa cho phép tổ chức các lớp **chia sẻ mã** chương trình chung, nhờ vậy có thể dễ dàng sửa chữa, nâng cấp hệ thống.
 - Định nghĩa sự tương thích giữa các lớp, nhờ đó ta có thể chuyển kiểu tự động.



Sharing is learning

6. Kế thừa

- Cú pháp khai báo:

```
Class Derived : <từ khóa dẫn xuất> Base{  
    ...  
};
```

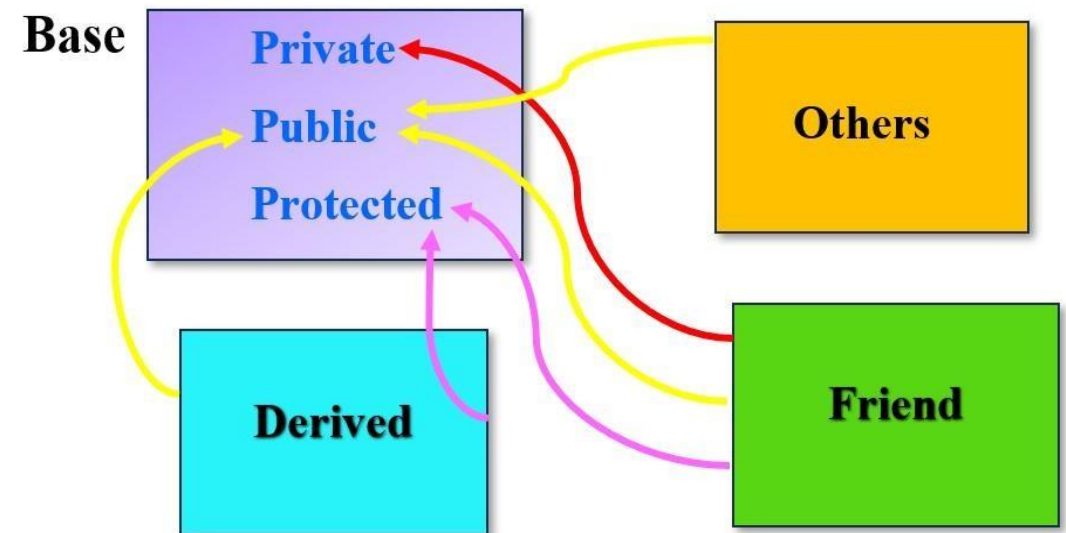
- Lớp Derived kế thừa từ lớp Base. Derived là lớp con (lớp dẫn xuất), Base là lớp cha (lớp cơ sở).
- Lớp con sở hữu các **thuộc tính, phương thức** của lớp cha mà không cần phải khai báo.



Sharing is learning

6.1. Phạm vi truy cập protected

- Sự kết hợp giữa **public** và **private**:
 - Giống với **private**, các thành viên **protected** không thể được truy cập từ bên ngoài lớp.



Sharing is learning

6.2. Các kiểu kế thừa

- Từ khóa dẫn xuất dùng để **thay đổi mức độ truy cập** của các thành viên ở lớp cha khi chúng được kế thừa xuống lớp con:



Sharing is learning

6.2. Các kiểu kế thừa

<div>Từ khóa dẫn xuất</div> <div>Phạm vi lớp cơ sở</div>	Private	Protected	Public
Private	X	X	X
Protected	Private	Protected	Protected
Public	Private	Protected	Public

6.2. Các kiểu kế thừa

- Lớp con kế thừa kiểu **public** từ lớp cha thì các thành phần **protected** của lớp cha trở thành **protected** của lớp con, các thành phần **public** của lớp cha trở thành **public** của lớp con.
- Lớp con kế thừa kiểu **protected** từ lớp cha thì các thành phần **protected** và **public** của lớp cha trở thành **protected** của lớp con.
- Lớp con kế thừa kiểu **private** từ lớp cha thì các thành phần **protected** và **public** của lớp cha trở thành **private** của lớp con.



Sharing is learning

6.2. Các kiểu kế thừa

- Lớp con được sở hữu các thuộc tính **private** của lớp cha.
- Tuy nhiên lớp con **không thể truy cập trực tiếp** đến các thuộc tính này mà phải thông qua các **phương thức truy vấn, cập nhật** được thừa hưởng từ lớp cha.



Sharing is learning

6.3. Phép gán với con trỏ trong kế thừa

- Một biến con trỏ kiểu lớp cha có thể giữ địa chỉ của của một đối tượng thuộc lớp con.
- Tuy nhiên ngược lại thì sai.
- Ví dụ:

Cha cha;

Con con;

Cha* chaptr = &con; // Đúng

Con* conptr = &cha; // Sai



Sharing is learning

Các điểm kiến thức

1. Lớp và đối tượng
2. Tính trừu tượng hóa và đóng gói
3. Phương thức thiết lập, phá hủy
4. Thành viên static, hàm bạn, lớp bạn
5. Nạp chồng toán tử
6. Kế thừa
7. Đa hình



Sharing is learning

7. Đa hình

- Ngoài việc được thừa hưởng các thuộc tính và phương thức từ lớp cha, lớp con còn có thể **định nghĩa thêm** các thuộc tính, phương thức mới và **định nghĩa lại** một vài phương thức ở lớp cha.
- **Phương thức ảo** là những phương thức lớp cha muốn lớp con định nghĩa một **phiên bản riêng** của nó.



Sharing is learning

7. Đa hình

```
class Cha{  
public:  
    virtual void Xuat() {  
        cout << "Cha";  
    }  
};
```

```
class Con: public Cha{  
public:  
    void Xuat(){  
        cout << "Con";  
    }  
};
```



Sharing is learning

7. Đa hình

- Phân biệt giữa nạp chồng (overload) và ghi đè (override):
 - **Nạp chồng:** Các phương thức có **cùng tên** nhưng danh sách **tham số khác nhau**, kiểu dữ liệu trả về có thể giống hoặc khác nhau.
 - **Ghi đè:** Các phương thức **giống nhau hoàn toàn** về tên, danh sách tham số, kiểu dữ liệu trả về.



Sharing is learning

7. Đa hình

- Khi gọi một **phương thức ảo** thông qua một **con trỏ** kiểu lớp cha, tùy thuộc vào **kiểu dữ liệu** của đối tượng mà biến con trỏ đang giữ địa chỉ, phiên bản của phương thức ảo trong **lớp tương ứng** sẽ được thực hiện -> tính đa hình.



Sharing is learning

7. Đa hình

Cha base;

Con derived;

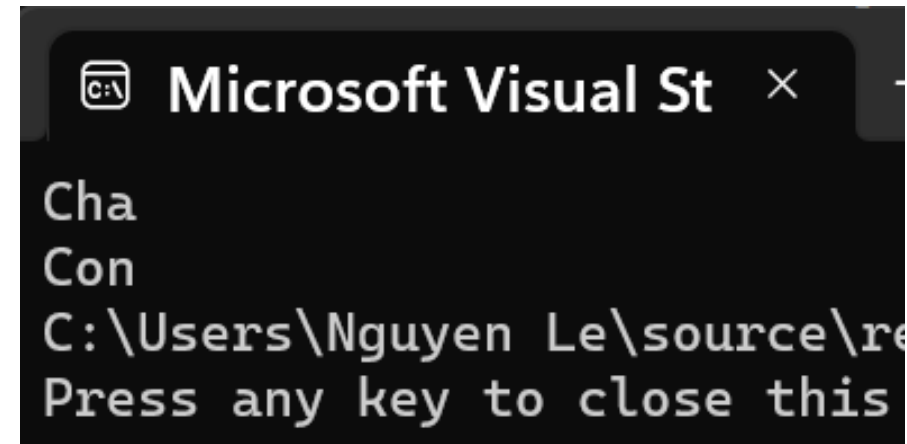
Cha* baseptr;

baseptr = &base;

baseptr->Xuat();

baseptr = &derived;

baseptr->Xuat();

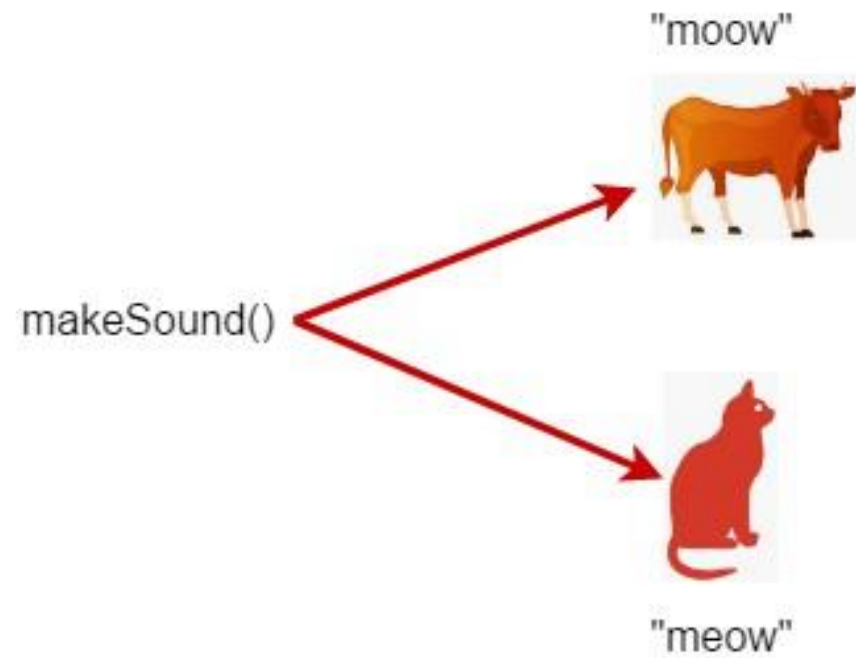


```
Microsoft Visual St x  
Cha  
Con  
C:\Users\Nguyen Le\source\re  
Press any key to close this
```



Sharing is learning

7. Đa hình



Sharing is learning

7. Đa hình

- **Hàm thuần ảo:**

Một hàm được tạo ra mà không có nội dung gì để phục vụ cho các lớp dẫn xuất kế thừa từ lớp cơ sở ghi đè lại.

- **Lớp trừu tượng:**

Một lớp có ít nhất một hàm thuần ảo được gọi là lớp trừu tượng.

Mục đích lớp trừu tượng được tạo ra là để các lớp dẫn xuất kế thừa lại



Sharing is learning

7. Đa hình

Ví dụ:

```
class Shape {  
public:  
    // phương thức thuần ảo:  
    virtual void draw() = 0;  
    // phương thức draw của Shape  
    // không có hành động cụ thể  
};
```



Sharing is learning

7. Đa hình

- **Lưu ý:** Khi gọi phương thức thông qua con trỏ lớp cha, ta chỉ có thể **gọi các phương thức mà lớp cha sở hữu**, cho dù con trỏ đang giữ địa chỉ của đối tượng thuộc lớp cha hay con.
- Ví dụ:

Con derived;

```
// phương thức do lớp Con định nghĩa kêu
```

```
derived.phuongThucCon(); // Đúng
```

```
Cha* baseptr = &derived;
```

```
baseptr->phuongThucCon(); // Sai!
```

7. Đa hình

- Có thể sử dụng **ép kiểu** để gọi các hàm của lớp con, tuy nhiên hành động này **dễ gây lỗi**, cần sử dụng cẩn thận.
- Ví dụ:

`((Con*)baseptr)→phuongThucCon(); // Đúng`



Sharing is learning

Đề 2021-2022 HKII

Thư viện trường học là trung tâm văn hóa, khoa học của nhà trường nhằm mở rộng kiến thức, góp phần nâng cao hiệu quả dạy - học và nghiên cứu khoa học của giáo viên và học sinh.

Một thư viện trường học cần quản lý danh sách các loại sách gồm 2 loại như sau:

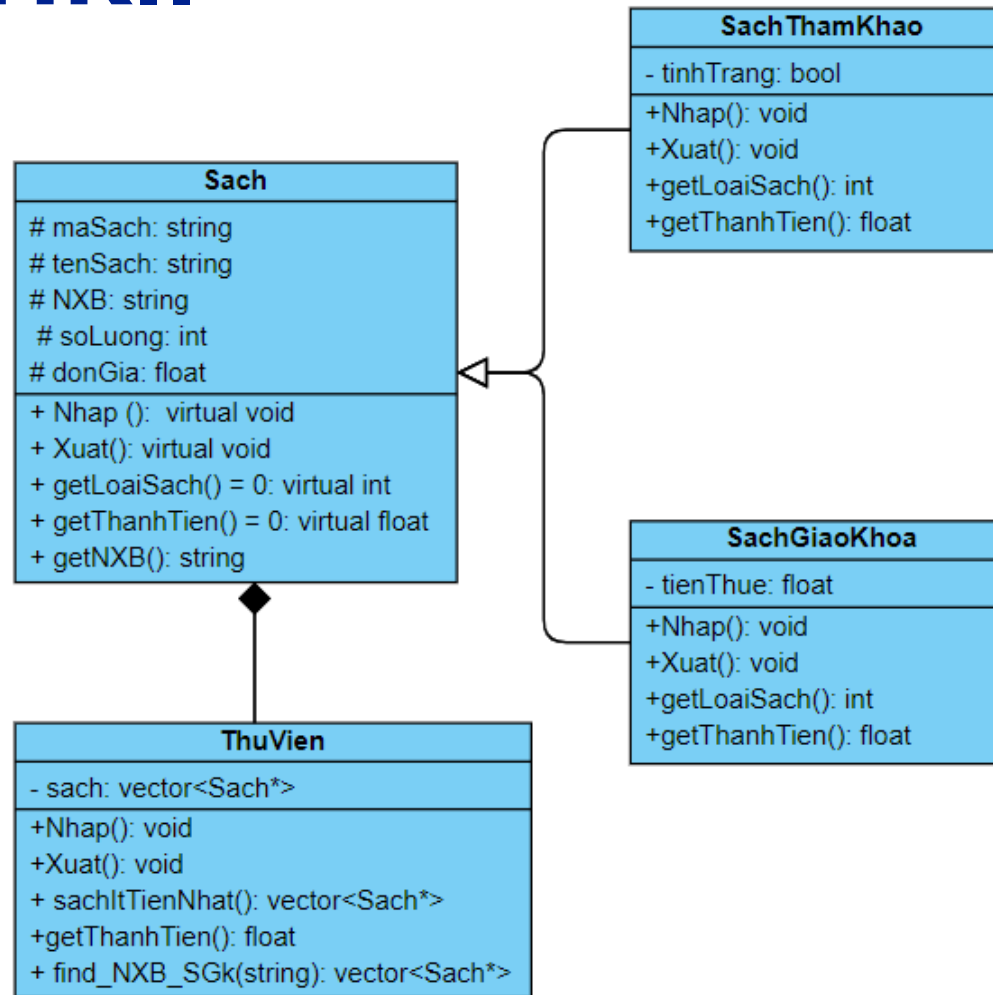
- **Sách giáo khoa:** mã sách, tên sách, nhà xuất bản, số lượng, đơn giá và tình trạng (0: cũ, 1: mới).
 - ❖ Nếu tình trạng sách là mới thì: Thành tiền = số lượng * đơn giá.
 - ❖ Nếu tình trạng sách là cũ thì: Thành tiền = số lượng * đơn giá * 50%
- **Sách tham khảo:** mã sách, tên sách, nhà xuất bản, số lượng, đơn giá và tiền thuế.
 - ❖ Thành tiền = số lượng * đơn giá + thuế

Đề 2021-2022 HKII

Yêu cầu:

1. **(1 điểm)** Áp dụng kiến thức lập trình hướng đối tượng (kế thừa, đa hình) thiết kế sơ đồ chi tiết các lớp đối tượng.
2. **(1 điểm)** Định nghĩa các lớp để quản lý theo yêu cầu của thư viện trường học.
3. **(3 điểm) Xây dựng các phương thức để thực hiện các yêu cầu sau:**
 - a. **(1 điểm)** Nhập và xuất danh sách các sách của thư viện trường học.
 - b. **(1 điểm)** Tính thành tiền mà thư viện trường học phải trả cho từng loại sách và hiển thị thông tin chi tiết những sách nào có thành tiền phải trả ít nhất.
 - c. **(1 điểm)** Tìm thông tin các sách giáo khoa của nhà xuất bản có tên do người dùng nhập vào.

Đề 2021-2022 HKII



Đề 2021-2022 HKII

```
class Sach{
    protected:
        string maSach;
        string tenSach;
        string NXB;
        int soLuong;
        float donGia;
    public:
        virtual void Nhap();
        virtual void Xuat();
        string getNXB();
        virtual int getLoaiSach()=0; //1.Sach giao khoa 2.Sach tham khao
        virtual float getThanhTien() = 0;
};
```

Đề 2021-2022 HKII

```
void Sach::Nhap(){
    cin.ignore();
    cout << "Nhap ma sach: ";
    getline(cin, maSach);
    cout << "Nhap ten sach: ";
    getline(cin, tenSach);
    cout << "Nhap nha xuất bản: ";
    getline(cin, NXB);
    cout << "Nhap số lượng sách: ";
    cin >> soLuong;
    cout << "Nhap đơn giá sách: ";
    cin >> donGia;
}
```


Đề 2021-2022 HKII

```
void Sach::Xuat(){
    cout << "Ma sach: " << maSach << "\tTen sach: " << tenSach
        << "\tNXB: " << NXB << "\tSo luong: " << soLuong
        << "\tDon gia: " << donGia;
}
string Sach::getNXB(){
    return NXB;
}
```

Đề 2021-2022 HKII

```
class SachGiaoKhoa: public Sach{  
    private:  
        bool tinhTrang;  
    public:  
        void Nhap();  
        void Xuat();  
        int getLoaiSach();  
        float getThanhTien();  
};
```

Đề 2021-2022 HKII

```
void SachGiaoKhoa::Nhap(){
    Sach::Nhap();
    cout << "Nhap tinh trang 0.Cu 1.Moi: ";cin >> tinhTrang;
}
void SachGiaoKhoa::Xuat(){
    Sach::Xuat();
    cout << "\tTinh trang: ";
    tinhTrang == true ? cout << "Moi": cout << "Cu";
    cout << "\tThanh tien: " << getThanhTien() << endl;
}
int SachGiaoKhoa::getLoaiSach(){
    return 1;
}
float SachGiaoKhoa::getThanhTien(){
    return (tinhTrang == true) ? soLuong*donGia : soLuong*donGia*0.5;
}
```

Đề 2021-2022 HKII

```
class SachThamKhao: public Sach{  
    private:  
        float thue;  
    public:  
        void Nhap();  
        void Xuat();  
        int getLoaiSach();  
        float getThanhTien();  
};
```

Đề 2021-2022 HKII

```
void SachThamKhao::Nhap(){
    Sach::Nhap();
    cout << "Nhap tien thue: "; cin >> thue;
}
void SachThamKhao::Xuat(){
    Sach::Xuat();
    cout << "\tTien thue: " << thue << "\tThanh tien: "
        << getThanhTien() << endl;
}
int SachThamKhao::getLoaiSach(){
    return 2;
}
float SachThamKhao::getThanhTien(){
    return soLuong*donGia + thue;
}
```

Đề 2021-2022 HKII

```
class ThuVien{  
    private:  
        vector<Sach*> sach;  
    public:  
        void Nhap();  
        void Xuat();  
        vector<Sach*> sachItTienNhat();  
        vector<Sach*> find_NXB_SGK(string s);  
};
```

Đề 2021-2022 HKII

```
void ThuVien::Nhap(){
    int n, lc;
    cout << "Nhap so luong sach thu vien quan li: "; cin >> n;
    sach.resize(n);
    for(auto& it : sach){
        cout << "Nhap lua chon 1.Sach giao khoa 2.Sach tham khao: ";
        cin >> lc;
        if(lc == 1) it = new SachGiaoKhoa;
        else it = new SachThamKhao;
        it->Nhap();
    }
}

void ThuVien::Xuat(){
    for(auto it : sach){
        it->Xuat();
    }
}
```

Đề 2021-2022 HKII

```
vector<Sach*> ThuVien::sachItTienNhat(){
    vector<Sach*> res;
    float Min = sach[0]->getThanhTien();
    for(auto it : sach){
        if( it->getThanhTien() < Min) Min = it->getThanhTien();
    }
    for(auto it : sach){
        if(it->getThanhTien() == Min) res.push_back(it);
    }
    return res;
}
```


Đề 2021-2022 HKII

```
vector<Sach*> ThuVien::find_NXB_SGK(string s){  
    vector<Sach*> res;  
    for(auto it : sach){  
        if( it->getLoaiSach() == 1 && it->getNXB() == s)  
            res.push_back(it);  
    }  
    return res;  
}
```

Đề 2021-2022 HKII

```
int main(){
    ThuVien thuvien;
    // Câu a
    thuvien.Nhap();
    thuvien.Xuat();
    // Câu b
    cout << "Sach it tien nhat\n";
    vector<Sach*> res1 = thuvien.sachItTienNhat();
    for(auto it : res1) it->Xuat();
    // Câu c;
    cout << "Nhap NXB sach tham khao muon tim: ";
    string s;
    cin.ignore();
    getline(cin, s);
    vector<Sach*> res2 = thuvien.find_NXB_SGK(s);
    if(!res2.empty()){
        cout << "Thong tin cach sach tham khao co ten NXB la: " << s << endl;
        for(auto it : res2) it->Xuat();
    }else cout << "Khong tim thay sach giao khoa co ten NXB la " << s << endl;
}
```

Sổ tay kiến thức OOP



Sổ tay kiến thức OOP



Sharing is learning

Form điểm danh



Sharing is learning

BAN HỌC TẬP CÔNG NGHỆ PHẦN MỀM

TRAINING CUỐI KỲ HỌC KỲ II NĂM HỌC 2023 – 2024



Sharing is learning

HẾT

**CẢM ƠN CÁC BẠN ĐÃ THEO DÕI
CHÚC CÁC BẠN CÓ KẾT QUẢ THI THẬT TỐT!**

 **BAN HỌC TẬP**

Khoa Công nghệ Phần mềm

Trường Đại học Công nghệ Thông tin

Đại học Quốc gia thành phố Hồ Chí Minh

 **CONTACT**

bht.cnpm.uit@gmail.com

fb.com/bhtcnpm

fb.com/groups/bht.cnpm.uit